AU/ACSC/2015

AIR COMMAND AND STAFF COLLEGE

AIR UNIVERSITY

Making IT Work

by

Shannon Moore

A Research Report Submitted to the Faculty

In Partial Fulfillment of the Graduation Requirements

Advisor: Dr. Fred Stone

Maxwell Air Force Base, Alabama

October 2015

Disclaimer

The views expressed in this academic research paper are those of the author and do not reflect the official policy or position of the US government or the Department of Defense. In accordance with Air Force Instruction 51-303, it is not copyrighted, but is the property of the United States government.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This study came about as a result of my curiosity during my assignment as the lead cost analyst for the Defense Enterprise Accounting and Management System (DEAMS). It was during this time on DEAMS that the Office of Management and Budget directed major automated information systems to deliver capability in smaller increments. It was also during this time that Congress cancelled the Expeditionary Combat Support System (ECSS).

Many of the software engineers from ECSS were moved to DEAMS. My conversations with them helped me understand that ECSS had followed senior leadership's direction and was broken into smaller, shorter increments, but the underlying software development methodology was not change. I wondered, "Could switching the software development methodology from Waterfall to Scrum have made a difference?"

Given the stove-pipe nature of Air Force (AF) information technology (IT) programs, I was not convinced that OMB was giving MAIS programs proper guidance. I was surprised to learn the full story behind the creation of the Waterfall methodology and the Department of Defense's wide-spread use of it. I was equally surprised to learn that Scrum is a viable software development methodology for AF IT programs because I initially thought it would take less time, but cost more money. Hopefully the knowledge presented in this work will help others to see how Scrum can prevent cost and schedule overruns for future AF IT programs.

I would like to thank my Professor, Dr. Fred Stone, for his guidance and candor. I have enjoyed the research process and am grateful for the opportunity to continuously improve my work. I would also like to thank my classmates for their comments, edits, and suggestions. Most importantly, I would like to thank my husband and children for your patience and support….Ahh TEAM!

ABSTRACT

The Office of Management and Budget (OMB) has instructed information technology (IT) programs to break projects into more manageable chunks of functionality that can be delivered every six to twelve months, like Scrum, an Agile software development method. OMB claims Agile methods, like Scrum, are necessary for the Air Force (AF) to successfully deliver IT systems on budget, on time, and with all of their planned capabilities. IT systems are critical to a more efficient and effective government. In the current fiscal environment, there are high expectations for IT to bring value to the American taxpayer and advance the mission of the warfighter. One IT program that was cancelled because it was not able to meet this expectation was the AF's Expeditionary Combat Support System (ECSS). In 2012, ECSS became the prime example of how an IT system designed to save billions of dollars could actually waste over a billion of taxpayer dollars without producing any usable capability.

Before ECSS was cancelled, OMB had concluded that the Waterfall software development methodology that ECSS used did not work. This research used an evaluative framework to determine if Scrum could have fixed Waterfall and delivered ECSS on time, within budget, and with all of its planned capabilities. Release 1 of ECSS was successfully broken into Scrum teams and reduced software development costs and schedule issues presented by Waterfall. Recommendations for future AF IT programs include use of Scrum tailored to meet Department of Defense-unique challenges.

INTRODUCTION

In 2003, Secretary of Defense, Donald Rumsfeld, "charged the services with six transformational objectives: Optimize support to the warfighter, Improve strategic mobility to meet operational requirements, Implement customer wait time as a cascading metric, Fully implement total asset visibility, Reengineer applicable processes and systems to increase overall communication, and Achieve best-value logistics while meeting requirements at reduced operating costs."[1] To meet the objectives, the Air Force (AF) appointed Grover Dunn, Deputy Chief of Staff for Logistics, Installations and Mission Support, as Director of Transformation to implement necessary changes to AF logistic business processes and the accompanying information technology (IT) systems.[2] The strategy that Dunn developed resulted in an AF campaign called Expeditionary Logistics for the 21st Century (eLog21).

The first enabling IT program designed to automate the eLog21 vision was the Expeditionary Combat Support System (ECSS). ECSS was supposed to integrate over 700 disparate logistic legacy systems into one platform, deliver $644M in savings by 2012, and $100M in savings every year thereafter.[3] In actuality, ECSS cost the American taxpayer more than $1 billion by November 2012 and was projected to cost over a billion more before yielding any significant capability in 2016.[4] ECSS should have delivered value to the American taxpayer by enabling the AF to better serve the warfighter.[5] ECSS ran over budget, behind schedule, and failed to deliver any usable capability to the field before it was cancelled in 2012.[6]

As early as 2010, OMB issued guidance to help IT programs, like ECSS, reduce the risk of multimillion dollar cost overruns and schedule delays and bring value to the American taxpayer. The guidance recommended the use of a software delivery consistent with an approach known as Agile, which calls for producing software in small, short increments.[7] However, Agile

methods like Scrum are not always suitable for every program and, at the time of this research, AF IT programs have been provided very little direction to help determine if Scrum can produce the anticipated outcomes.

If Scrum cannot prevent cost and schedule overruns, taxpayer dollars will continue to be wasted and critical capabilities like supporting the warfighter will not be met. This research examined, whether the Scrum methodology of Agile software development can prevent cost and schedule overruns for future AF IT systems. The research focused on the unique program characteristics of ECSS Release 1 and addressed the central issue of fixing cost and schedule overruns associated with large software development programs.

This research used an evaluative framework to determine if Scrum could deliver future AF IT systems on time, within budget, and with all of its planned capabilities. Before ECSS was cancelled, OMB had concluded that the Waterfall software development methodology ECSS used did not work. Therefore, Release 1 of ECSS was broken into Scrum teams to see if it reduced, did nothing, or increased software development costs and schedule issues presented by Waterfall.

Literature Review

This analysis focused on these documents: *Department of Defense Military Standard 2167* (DOD-STD-2176) (1985), *25 Point Implementation Plan to Reform Federal Information Technology Management* (2010), *Business Capability Life-cycle (BCL) guidance* (2010), and *DoD Instruction 5000.2* (2015). These unclassified documents were collected from the Air Force Life Cycle Management Center's (AFLCMC) document repository. Each document was reviewed to identify statutory and mandatory processes that would have influenced the structure and content of ECSS' software development from 2004 to 2012. Due to the high failure rate in applying the mandatory processes in DOD-STD-2176, a Defense Science Board Task Force

report on military software was also reviewed to determine what, if any, recommendations may have applied to ECSS. The documents dated before 2015, also provided a chronological time line for understanding DoD's shift to requiring software development in smaller, more Agile, increments. DoD Instruction (DoDI) 5000.2 was used to establish an understanding of current software development requirements that will impact future AF IT programs.

The following ECSS literature was reviewed: an Institute for Defense Analyses (IDA) report, fact sheets, a permanent subcommittee on investigations report, and various internet articles. The October 2011 IDA report provided root cause analysis of cost and schedule overruns related to data readiness, a deliberate delivery strategy, requirements increase, and schedule delays. This research utilized IDA report information on the deliberate delivery strategy, which identified the results of restructuring the program into smaller increments, and requirements increase. ECSS facts on planned acquisition milestones and dates, functional capabilities by release, contracts and contractors, and life-cycle costs were derived from the IDA report, fact sheets, and permanent subcommittee on investigations report.

The following Waterfall, Agile and Scrum methodology literature was reviewed: "Managing the Development of Large Software Systems," *Scaling Software Agility Best Practices for Large Enterprises* from the Air Force Institute of Technology library, print and internet articles from the Muir S. Fairchild Research Information Center, and YOUTUBE videos from the World Wide Web. The 1970 article "Managing the Development of Large Software Systems" provided an in-depth overview of the steps and risks associated with the Waterfall methodology.[8] The book *Scaling Software Agility Best Practices for Large Enterprises* provided a detailed description of why the Waterfall methodology failed in the past and offered an understanding of how Agile directly addressed the causes of the failures.[9] Print and internet

articles were limited to those written by Jeff Sutherland and Ken Schwaber, two of the founders of the Agile Manifesto and co-creators of the Scrum methodology. Similarly, a series of YOUTUBE videos published by Open View Venture Partners featuring Jeff Sutherland were previewed to gain an understanding of how Scrum was created and the basics on how it works.

Results of the Literature Review

"Managing the Development of Large Software Systems" has seven implementation steps of the Waterfall methodology and the five fixes required to eliminate risks associated with executing a sequential process.[10] For the purposes of this research, a sixth fix identified by the author as required for risk reduction of a Waterfall software project titled Iterative Interaction Between Phases was included. The six fixes: 1) Program Design Comes First, 2) Document the Design, 3) Do It Twice, Plan, 4) Control and Monitor Testing, 5) Involve the Customer, and 6) Iterative Interaction Between Phases served as the evaluation criteria for determining if Scrum can prevent cost and schedule overruns for future AF IT programs.[11]

There are twelve principles of Agile and the Agile Manifesto. Scrum literature provided the definitions of the three main roles for Scrum: Product Owner, Scrum Master, and Development Team. This research used Jeff Sutherland's description of the Scrum process from YOUTUBE coupled with first-hand knowledge of the process to explain how Scrum works.[12] The components of the Scrum process: Scrum Team, Product Backlog, Sprint Backlog, Sprint Planning Meeting, Daily Scrum, Burndown Chart, Sprint Review, and the Sprint Retrospective were applied against the fixes for Waterfall.

The analysis was used to determine the results of using Scrum on ECSS Release 1, post-critical change. The ECSS literature review yielded that Release 1, post-critical change, was planned for 26 months and was to produce 250 custom Reports, Interfaces, Conversions, and

Extensions (RICE) objects.[13] As stated in 10 U.S.C. Ch 144A, a critical change was determined when a major automated information system (MAIS) program failed to "achieve a Full Deployment Decision (FDD) within five years after funds were first obligated for the program."[14] A critical change triggered a formal technical risk assessment of the program by the Office of the Secretary of Defense, which typically resulted in a recovery plan that tailored the program's acquisition strategy and updated the milestone schedule with new success criteria.[15]

The five year time line for ECSS began on 31 August 2005, when the Milestone Decision Authority (MDA), approved Milestone (MS) A and program funds were first obligated to stand up the program office at Wright-Patterson Air Force Base, not in September 2006 when the software development contract was awarded.[16] The MAIS Quarterly Report dated 31 October 2010, highlighted that the MDA had not approved a FDD for ECSS by 31 August 2010, and determined that the program had experienced a critical change.[17]

BACKGROUND

With the introduction of the main frame computer in the mid-1950s, the primary methodology used to produce software was the code and fix formula.[18] While remarkably successful, the government and some corporate organizations did little or no initial planning.[19] For example, the accepted practice for software developers was to receive a request, start developing, which could take up to one year, and then make the fixes as they occur, which could take as many as five years or more.[20] If major architectural changes resulted from a fix, large portions of the code had to be re-written.[21]

As a result, the 1960s ushered in the additional steps of design before coding, and test and maintenance after coding. These added steps were well received by the software community,

and not long after, the formal software development life cycle (SDLC) emerged to establish a solid framework for building software.[22] The SDLC contained five core activities: Requirements Analysis, Design, Implementation, Testing, and Evolution or Maintenance (Figure 1).

Figure 1. The Software Development Life Cycle (SDLC).
(Reprinted from "The Software Development Life Cycle - Overview", http://www.stylusinc.com/BI/ it-outsourcing/the-software-development-life-cycle-sdlc/)

SDLC activities are part of an iterative process. Thus, the most profound and popular methodology for software development *should not* have been a linear, sequential process.

Waterfall

In 1970, Winston Royce, software development pioneer, wrote "Managing the Development of Large Software Systems."[23] The term for the sequential process, Waterfall, was never used by Royce.[24] In his article, Royce portrayed a total of ten figures, where each figure built upon the concepts of the previous one and advanced the ideas within his writing.[25] Experts fixated only on figure two, which portrayed software development as a sequence of seven implementation steps – System Requirements, Software Requirements, Analysis, Program Design, Coding, Testing, and Operations (Figure 2).[26]

Royce's article was actually more of a cautionary tale against using the sequential process depicted in figure two, than an endorsement for using it. For instance, Royce said, "I believe in this concept, but the implementation described above is risky and invites failure."[27]

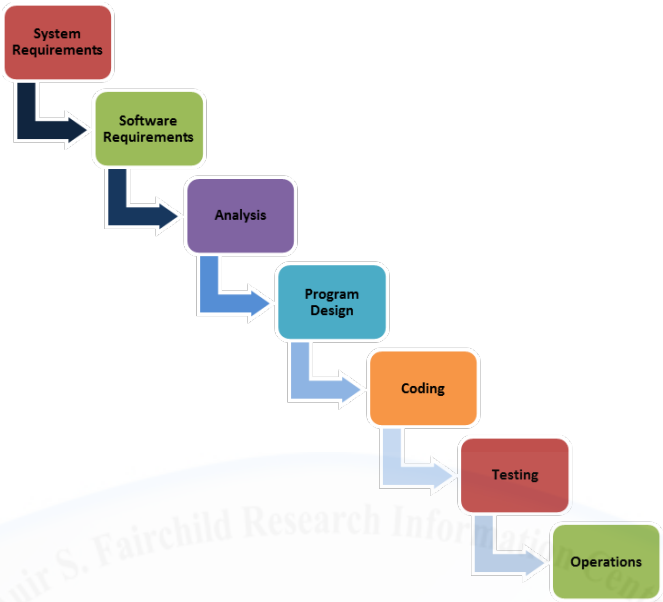

Figure 2. The Waterfall Model.
(Adapted from Royce, Dr. Winston W., "Managing The Development of Large Software Systems", *Proceedings, IEEE WESCON* (August 1970): 329)

In theory, the phases of the sequential process flow continuously from one step to the next, like a waterfall. However, in practice, as Royce cautioned, its primary disadvantage is that it is not known how the system will *actually* work until the testing phase.[28] Consequently, if testing fails, the necessary changes will return the development process back to program design or possibly software requirements, resulting in "up to 100-percent overrun in schedule and/or costs."[29]

Although Royce added iterative interaction between steps, documentation requirements, and critical reviews to his methodology, he acknowledged that he believed the approach illustrated in figure two was fundamentally sound, and with that, experts coined the methodology Waterfall. By narrowing Waterfall to figure two, additional steps that Royce felt must be added

in order to eliminate most of the cost and schedule development risks were disregarded.[30] If the additional steps - Program Design Comes First; Document the Design; Do It Twice; Plan, Control and Monitor Testing; Involve the Customer; and Iterative Interaction Between Steps – had been included, the resulting methodology would never have been considered a sequential approach. In addition, it may not have received wide-spread use throughout the software development community.

Why the Waterfall Methodology Did Not Work for ECSS

The DoD is partly responsible for the wide-spread popularity and use of the Waterfall methodology. In 1985, DoD released Military Standard 2167 (DOD-STD-2167), *Defense System Software Development*, which mandated software development programs to follow a uniform process that mirrored the Waterfall methodology. DOD-STD-2167 incorporated best practices from DoD and the software industry that had demonstrated cost-effectiveness. Referred to as the "system development cycle within the system life cycle", DOD-STD-2167 paralleled the seven implementation steps of Waterfall with detailed implementation steps for software requirements analysis, preliminary design, detailed design, coding and unit testing, integration and testing, and configuration testing.[31] The only distinction between DOD-STD-2176 and Waterfall was the addition of six software engineering reviews, one after each implementation step (e.g., system requirements review, preliminary design review, critical design review), where the government approved the system integrator's (SI) work before moving on to the next step.

Less than two years after initiation, DoD had experienced significant cost and schedule problems in applying DOD-STD-2176 and convened a task force chaired by Dr. Frederick Brooks, a software engineering expert, to assess the problem.[32] The task force concluded that

the problems in military software development were not technical problems, despite supportive evidence that pointed to technical deficiencies, but managerial problems. For example, the report stated that major changes to "attitudes, policies, and practices concerning software acquisition" were needed.[33] However, in the body of the report the task force described substantial issues with DOD-STD-2167's Waterfall approach, citing that requirements definition was a difficult, yet critical part of the software development process that required iteration.[34] The task force recommended iterative and evolutionary development, referred to today as incremental development, to correct the cost and schedule issues experienced with DOD-STD-2167.[35] However, the task force left the implementation of the recommendations unresolved; citing that the changes needed would create chaos with the DoD acquisition process for competitive procurement of IT software development services.[36]

Agile

Software Development methodologies introduced in the 1980s, such as Spiral, and 1990s, such as Rapid Application Development, adapted Waterfall with one or more of the original five fixes Royce introduced. However, it would not be until the late 1990s - early 2000s, that an alternative to Waterfall called Agile would be introduced.

Agile was a collection of software development methodologies such as Extreme Programming, Adaptive Software Development, Crystal, and Scrum that were used to implement the SDLC and offered an alternative to traditional methods of developing software.[37] There were twelve principles for the Agile methodology alternative, with the highest priority being customer satisfaction through software that was planned iteratively and implemented by an empowered team.[38]

Six principles were dedicated to switching from predictive to adaptive, or iterative, planning. Predictive planning methodologies, like Waterfall, implemented the work in phases according to a plan. Founders of the Agile methodology recognized this approach created an unrealistic dependency on stable requirements in the early phases of a program in order to stay within budget and on schedule. As a result, Agile methodologies offered a process that welcomed changes in requirements and delivered working, sustainable software in small increments.[39]

Five principles were dedicated to forming an empowered team. Founders of the Agile methodology emphasized allowing the people assigned to the project to decide the process they would follow instead of upper management directing the tasks and deadlines.[40] This notion to get a good group of people together and let them decide the process underpinned what the founders called the Agile Manifesto.

In 2001, seventeen modern-day software pioneers gathered together to discuss the future of software development.[41] The meeting resulted in a philosophy that formalized the principles discussed above. The group stated that their experiences had led them to place importance on processes that valued:

> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan[42]

Immediately following these values, the group added a sentence: "That is, while there is value in the items on the right, we value the items on the left more."[43] The group placed the words on the left in bold font, acknowledging that the words on the right are necessary for software development, but not the most important for Agile development.

Scrum

Jeff Sutherland and Ken Schwaber, two of the seventeen authors of the Agile Manifesto, co-developed Scrum in 1995. Scrum is the leading Agile development methodology for implementing large and complex software projects.[44] Scrum has been widely used in the public and private sector and is one of the two methodologies recommended by the Government Accountability Office (GAO) for use in implementing federal IT projects.[45]

How Does Scrum Work?

Scrum has three main roles: Product Owner, Scrum Master, and the Development Team.[46] The Product Owner gathers input from end-users, customers, team members, and other stakeholders to establish a list of features and business requirements called the Product Backlog. There is only one Product Owner and only this individual can add, update, delete, and prioritize the backlog.[47] The Product Owner attends all meetings and has the ultimate say in whether or not to cancel development before the software is complete.[48] The Scrum Master documents the tasks and activities of the team, but is not a traditional project manager. Instead, the Scrum Master's role is to serve the Product Owner and Development Team.[49] This individual is trained on how to guide the team through the Scrum process and protect the team from external disturbances. The Scrum Master also attends all meetings. The Development Team consists of seven, plus or minus two people.[50] If a project is large, multiple Development Teams can be formed, but each should not exceed nine people. The team is cross-functional with members representing each phase of the development process, which includes the customer/requirements, software design, programming, and test.[51] The team is self-empowered, meaning the group determines how it will be organized and managed.[52] Each member of the Development Team is

co-located and attends all of the meetings. The combination of the Product Owner, Scrum Master, and the Development Team establish the Scrum Team.

The Scrum Team works in Sprints to deliver in increments and reports progress using a Burndown chart.[53] A Sprint is a fixed period of time, typically two to four weeks, where the team pledges to develop a segment, or increment, of capability.[54] The team builds the software incrementally in Sprints to prevent burnout, increase moral, and encourage continual learning. The aim of the Sprint is to produce 100 percent of what the team has committed to and demo working software at the end.

Sprint has four types of meetings – planning, daily, review, and retrospective.[55] The Sprint Planning meeting is fixed to a maximum of eight hours at the start of the Sprint.[56] In this meeting the team selects a manageable set of functionality/capabilities from the Product Backlog and places it into the Sprint Backlog.[57] During this meeting, the team makes a commitment on how much they will produce and determines the length of the Sprint. As a self-empowered team, the group creates a resource-loaded task list that ensures everyone on the team participates, regardless of their experience level.

The second type of meeting, the Daily Scrum, resulted from Sutherland's research on Bell Laboratories (Labs). Bell Labs had observed that in order to build software fast, the people responsible for building it could not be in meetings that consumed large portions of their work day. Bell Labs reported that the highest performance teams were driven by daily meetings no more than 15 minutes in length where everyone stood and faced each other, like a huddle in a football game, and had an opportunity to contribute.

Sutherland and Schwaber designed the Daily Scrum meeting with these characteristics in mind, and added that non-team members could attend, but only members of the Scrum Team

could talk and ask questions.[58] In the Daily Scrum meeting each team member provides the following information: what was completed yesterday, what will be completed today, and where is help needed.[59] The purpose of the Daily Scrum is to ensure that everyone on the team knows what others are doing, understands what is going to be done next, and how collaboration can help move the team forward faster. This information also helps the Product Owner determine if the Sprint should continue or be canceled.

The Scrum Master takes the information from the Sprint Planning and Daily Scrum meetings and updates the Burndown chart. Sutherland developed the reporting within the Daily Scrum meeting as an improvement to a long-standing problem with reporting using Gannt charts. A Gannt chart was commonly used by software project managers to show planned tasks and events. With a predictive planning methodology where the requirements are stable, the Gannt chart presumably worked well; however, with an adaptive process framework, like Scrum, the chart proved difficult to use and update. Sutherland pulled from experience as a fighter pilot, and developed the Burndown chart with characteristics similar to landing an airplane. The Burndown chart showed how fast the Scrum team was going, how much still had to be done before "touchdown", and visually indicated whether or not the team was going in the right direction. The Burndown chart is placed on the wall where the Daily Scrum meeting is held so that everyone can see the Sprint Backlog status, tasks, and who is responsible.

The third type of meeting is called the Sprint Review meeting where everyone on the Scrum Team and end-users come together for no more than four hours to demo the potentially working product increment (Figure 3).[60] The definition for a working product increment is functionality that has been designed, built, fully tested, and has no major defects. The importance of including the end-user in this meeting is to gather feedback on the software.

When using the Scrum methodology, learning from the first Sprint is passed along to the next Sprint. After the Sprint Review, a Sprint Retrospective meeting, not to exceed three hours, is held where only members of the Scrum Team discuss what went wrong and what went right with the Sprint.[61]

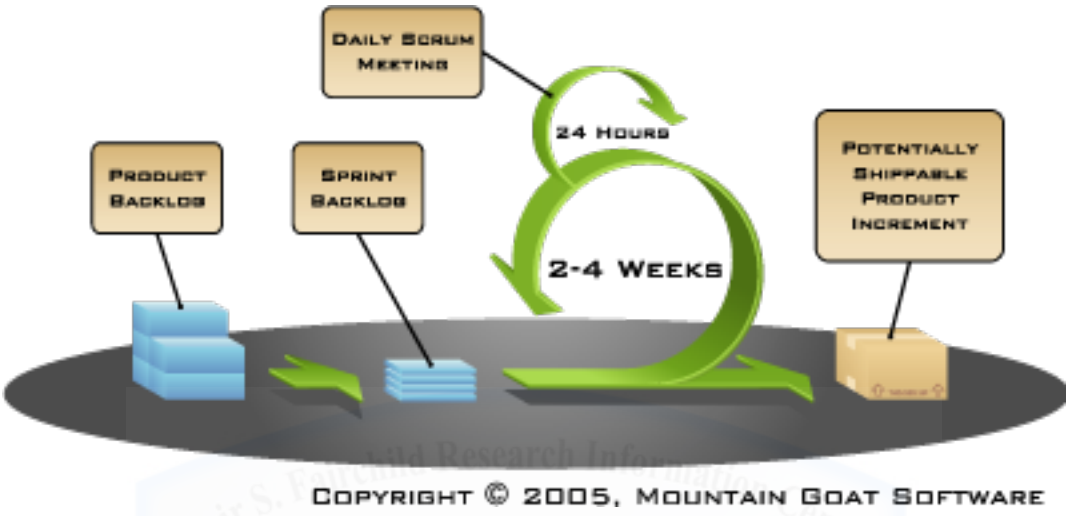

Figure 3. Scrum Development Lifecycle.
(Reproduced with permission from https://www.mountaingoatsoftware.com/agile/scrum/images)

During the Sprint, the Product Owner revises the Product Backlog to reflect new or modified requirements, as well as arranges the backlog in the right order so when the final product is done it will be exactly what the customer envisioned. Upon completion of the Sprint Retrospective meeting, the Scrum Team returns to the next Sprint Planning meeting to start the next Sprint.

EVALUATION RESEARCH CRITERIA AND ANALYSIS

This research evaluated the key aspects of the Scrum methodology against Royce's additional features that must be added to Waterfall in order to eliminate most of the cost and schedule risks experienced with the methodology.[62] The additional features were: 1) Program

Design Comes First, 2) Document the Design, 3) Do It Twice, 4) Plan, Control and Monitor Testing, and 5) Involve the Customer.[63] Although Royce did not identify it as an additional step Iterative Interaction Between Phases was added to the list of additional features because Royce mentioned it as necessary to prevent cost and schedule risks. This research summarized Royce's main ideas related to the selected criterion below.

Iterative Interaction Between Steps

The seven implementation steps for software development – system requirements, software requirements, analysis, program design, coding, testing, and operations - should not be confined to successive steps. By contrast, there should be iterative interaction between the preceding, current, and successive steps (Figure 4). As each implementation step progresses the development of the software toward the operations step, there may be a need to return to the preceding step. Limiting the iteration back one step reduces the scope of the change process down to a manageable amount. For example, an issue discovered in test could initiate a change to coding, but not a change to program design.

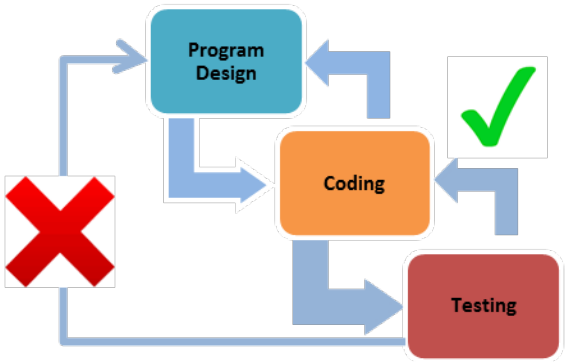

Figure 4. Iterative Interaction Diagram.
(Adapted from Royce, Winston, "Managing The Development of Large Software Systems", *Proceedings, IEEE WESCON* (August 1970): 330)

Once the Scrum Team pulls requirements from the Product Backlog into the Sprint Backlog, the Sprint commences a two to four week cycle with iterative interaction between four of the seven implementation steps: analysis, program design, coding, and testing.

The first two Waterfall steps associated with system and software requirements can only be revisited by the Scrum Team once the Sprint is complete. This eliminates the risk of up to 100 percent of cost and schedule overruns that Royce identified because it prohibits changes in new or modified requirements from interrupting the development of the software. The Sprint deliverable (i.e., increment or release) does not reach the operations step until after the Sprint Review is successfully completed. Operations cannot return to the preceding step, test, but returns the development process back to the Product Backlog.

Program Design Comes First

In order to address the infrastructure needs of the software, Royce added an additional step, preliminary program design, between software requirements and analysis (Figure 5). When using the Waterfall model, the program manager gathers system and software requirements and provides them to the analysts and programmers to complete the analysis step of the process. The addition of a preliminary program design step before analysis requires a program designer to design the infrastructure of the system before the analysts and programmers design the end-user interface. For example, preliminary program design equates to drawing blue prints with a plumbing layout for an entire home before picking out what brand of toilet or faucet would be installed. In the case of Waterfall, the program designer creates a detailed system overview that includes the following infrastructure designs and definitions: database and database processors, data storage, execution timing, and operating procedures. This ensures that all of the program

designers, analysts, and programmers understand the system and can contribute to the design process.

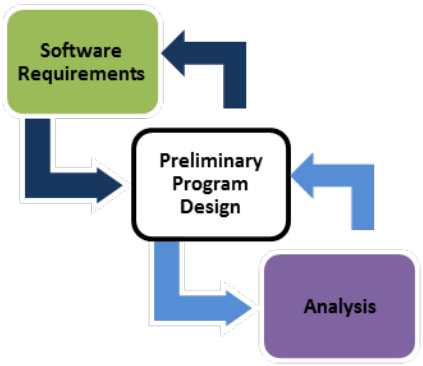

Figure 5. Add Preliminary Program Design.
(Adapted from Royce, Winston, "Managing The Development of Large Software Systems", *Proceedings, IEEE WESCON* (August 1970): 331)

Scrum did not address this fix in its entirety. Royce's fix was to design the entire infrastructure before analysis began. With Scrum everything is incremental, including the infrastructure design. The Scrum Development Team would design the infrastructure (database, database processes, execution timing, and procedures) for the immediate Sprint only.

Document the Design

High quality documentation is needed to produce a quality software product. The documentation is used to communicate the status of the development effort to the customer, management, stakeholders, and other technical experts on the development team. A total of six documents are produced. Document number one, software requirements, and document number two, preliminary design specifications, are produced before analysis in the software requirements and preliminary program design phases. The program design phase yields three documents: interface specifications (document three), final design (document four), and test plan

specifications and results (document five). The sixth document, operating instructions, is produced during the operations phase.

As outlined above, documentation is required for several phases of the process; however, the quality of the documentation is not apparent until the testing and operational phase. Table 1 summarizes how the quality of the documentation can affect the testing, operational, and post-operational phases.

	Quality Documentation	Poor Documentation
Testing Phase	Any technical expert assigned to the team can fix an issue identified during testing.	Only the technical expert that designed or programmed the mistake can fix it because only he/she understands it.
Operational Phase	Less expensive personnel can be trained on how to operate and maintain the software system.	The software must be operated and maintained by the higher paid personnel that built it.
Post-Operational Phase	Facilitates real-time redesign, updating, and retrofitting of the software in the field.	The infrastructure of the operating software must be reworked and redeployed or possibly scrapped altogether.

Table 1. Quality vs. Poor Documentation Summary.
(Adapted from Royce, Winston, "Managing The Development of Large Software Systems", *Proceedings, IEEE WESCON* (August 1970): 332)

The main purpose of the documentation is to communicate status to the customer, management, stakeholders, and other technical experts on the development team. The Burndown chart used by the Scrum Team does an outstanding job of communicating status; however, it is not a fully documented set of specifications and instructions for the software. A secondary purpose of the documentation is to provide new development team members, testers, or operations personnel with an in-depth understanding of the software. The in-depth understanding reduces the risk of cost and schedule delays because time is not needed to figure out what the original designer/programmer intended. Therefore, Scrum's lack of any formal documentation is unsatisfactory to address the risk posed by not having in-depth documentation.

Do It Twice

In order to arrive at an error-free software program, a pilot effort simulating the final product is built in advance of the first operational deployment of the software release. The pilot is condensed and should take approximately one-fourth to one-third of the amount of time to develop. For example, if the software release is planned for twelve months, the pilot should take three months to develop.

The pilot, which consists of abbreviated preliminary design, analysis, program design, coding, testing, and usage phases, is developed in parallel with the software release's preliminary program design phase.[64] The results of the pilot's usage phase are used to trouble-shoot the release design and scope down infrastructure requirements.

A Sprint is equivalent to a pilot of a software release. Within a Sprint, the software is tested incrementally, usually weekly, as the software is developed. The software product reviewed during the Sprint Review meeting for acceptance may be the third or fourth prototype of the Sprint. Additionally, the Sprint Retrospective meeting where the Scrum Team discusses what went right and what went wrong aides in reducing the amount of re-work for future Sprints.

Plan, Control and Monitor Testing

Test is "the phase of greatest risk in terms of dollars and schedule."[65] All of the previous fixes that have been described above contribute to uncovering and solving issues before entering the test phase. Beginning in the program design phase, the plan for testing is created. During testing, independent test specialists that did not contribute to the software design should administer the tests. This ensures that someone that did not participate in creating the design and/or documentation can use the software features according to the instructions and produce the expected results. Today this is referred to as user-friendly software.

The test phase is controlled such that every interface is visually inspected for design flaws before beginning a test, and every logic path is checked during the execution of the test script.[66] Once the logic path is checked it is documented and certified as error-free. The test phase is monitored to ensure that test standards and procedures are adhered to and tools, like computers, local area networks, and wide area networks, are working properly and do not impede testing.

The Scrum Development Team includes a technical writer and a test specialist. The Sprint Planning meeting identifies what requirements would be pulled into the Sprint Backlog. The technical writer develops test scripts to test the functionality of the requirements. The test specialist executes the test scripts according to the planned test event for the Sprint. The designers, analysts, and programmers monitor the execution of the test script. As mentioned previously, software within a Sprint is usually tested on a weekly basis. The numbers of test events are determined by the length of the Sprint. For example, a two week Sprint has two test iterations before the Sprint Review for final acceptance.

Involve the Customer

Beginning with system requirements, involvement of the customer is "formal, in-depth, and continuing."[67] The customer generates and updates system requirements and participates in software reviews following the design and analysis phases. The customer provides key personnel to serve as independent test specialists. Most importantly, the customer approves the final software acceptance review.

The Product Owner is the customer and is an essential part of the Daily Scrum meeting. For more complex software Sprints, the customer is also a key member of the Scrum

Development Team. The customer is co-located with the Scrum Team and involved in each iterative phase of the Sprint.

To summarize the analysis of Scrum, Royce felt each of the additional fixes/steps were necessary to "transform a risky development process into one that will provide the desired product."[68] Based on the analysis of this research, the Scrum methodology has the potential to improve the Waterfall process.

RESULTS

This research used an evaluative framework to determine if Scrum could deliver future AF IT systems on time, within budget, and with all of its planned capabilities. This research focused on the unique program characteristics of ECSS and addressed the central issue of fixing cost and schedule overruns associated with large software development programs.

Results of Criteria Applied to ECSS

The results of the analysis above were applied to ECSS by using historical data from the Air Force Cost Analysis Agency (AFCAA) and AFLCMC standards for MAIS programs to determine two outcomes. First, would use of the Scrum methodology for developing ECSS Release 1, post-critical change, deliver the release on time and with of its planned capabilities? Second, would using the Scrum methodology deliver ECSS Release 1 within budget?

On Time and With Planned Capabilities

ECSS Release 1, post-critical change, was planned for 26 months and was to produce 250 RICE Objects.[69] According to the November 2010 Release 1 schedule, development started in April 2010 and ended in March 2012. There were three pilots planned (A, B, and C) within the release before fielding began in June 2012.

Historical data for MAIS programs collected by the AFCAA shows that it takes 922 hours, regardless of software development methodology, to plan/analyze, design, develop, and test one RICE object.[70] For ECSS Release 1 to produce 250 RICE objects it would take 230,550 hours.

Hours per RICE Object		ECSS Release 1 RICE Objects		Total Hours to Develop RICE Objects
922 Hours	*	250 Objects	=	230,550 Hours

The AFLCMC standard man-month and maximum hours for a Sprint are equal. The AFLCMC standard for one man-month is 160 hours, or 40 hours per week. The maximum duration for a Sprint is four weeks.

AFLCMC Standard Man-Month		AFLCMC Standard Man-Month Hours per Week		Maximum number of weeks in a Sprint
160 Hours	=	40 Hours	*	4 Weeks

The AFCAA recommended maximum staff size for a SI is 100 full-time equivalents (FTE). Using the maximum Scrum Development Team size of nine FTEs, there would be five SI FTEs on a team: one designer, two analysts, and two programmers. The remaining four FTEs would be Government FTEs: two functional requirement SMEs and two testers. The maximum number of Scrum Development Teams is 20.

Maximum SI Staff Size		SI FTEs per Scrum Development Team		Maximum number of Scrum Development Teams
100 FTEs	÷	5 FTEs	=	20 Teams

Using the Scrum methodology as described in ***How Does Scrum Work***, it would take 20 Scrum Development Teams a total of eight months to deliver 250 RICE objects to the field.

Eighteen months shorter than the post-critical change schedule, and well within the 18 to 24 month requirement established by DoDI 5000.02 for incrementally deployed software intensive programs.

Maximum number of Scrum Development Teams		Scrum Development Team Hours per Month (9 FTEs * 160 Hours)		Total Scrum Development Team Hours Per Month
20 Teams	*	1,440 Hours	=	28,800 Hours Per Month
230,550 Hours ÷ 28,800 Hours per Month = 8 Months				

Using the Scrum methodology for developing ECSS Release 1, post-critical change, would deliver the release on time and with of its planned capabilities.

Within Budget

Adopting Scrum requires additional costs for: Scrum training for Government Scrum Team members and a co-located physical environment for the Product Owner, Scrum Master, and Scrum Development Teams. A two-day training course that covers the mechanics, roles, principles and process of Scrum costs approximately $1,500 per person.[71] For the 20 Scrum Teams, the Government would need to train 80 people on the basics of Scrum. Additionally, one person would receive the Product Owner training and 20 people would receive the Scrum Master training. The Product Owner training is also $1,500 and the Scrum Master training is $1,495.[72] The minimum Government investment for Scrum training is $151,400.

Scrum Mechanics Training for 80 FTEs		Scrum Product Owner Training for 1 FTE		Scrum Master Training for 20 FTEs		Minimum Government Investment for Scrum Training
$120,000	+	$1,500	+	$29,900	=	$151,400

Facility costs are based on historical data from commercial rental space in Dayton, Ohio for 200 people. The annual cost of $265,456 includes: chairs, cubicles, conference tables, conference chairs, bookcases, file cabinets, telephones, desktop computers, monitors (2 per desk), keyboard and mouse peripherals, projector, break room equipment, conference phone, multi-function printer/copier/scanners, fax machine, shredder, local area network servers/routers/switches, and network cables. The facility would be leased for a minimum of one year.

The AFLCMC historical hourly labor rate for SI RICE development is $140.34. The monthly cost for 100 FTEs is $2,245,440. The total cost for eight months is $17,963,520.

SI RICE Development costs ($140.34 * 100)		AFLCMC Standard Man-Month		SI RICE Development cost per Month
$14,034	*	160 Hours	=	$2,245,440
$2,245,440 * 8 Months = $17,963,520				

Per AFLCMC historical data for program management for MAIS programs, the Government would expect to pay approximately 25 percent, or $4,490,880, of the labor total for general administration and overhead (G&A) on the SI contract. Table 2 summarizes the combined costs for Scrum training, facilities, SI labor and G&A; the expected costs for software development are $22, 871,256.

Item	Costs in Millions
Scrum Training	$ 151,400
Facilities	$ 265,456
SI Labor for Software Development	$ 17,963,520
SI G&A	$ 4,490,880
Total Costs	**$ 22,871,256**

Table 2. Total Costs Using Scrum

The total costs represent four percent of the original SI contract award of $628 million. However, it is important to note that the original ECSS SI contract included costs for four releases, change management, and training. At the time of this research, these costs were not separately identifiable. Nonetheless, based on the results above it is fair to infer that using the Scrum methodology would deliver ECSS Release 1 within budget.

RECOMMENDATIONS

This research supports OMB and GAO's recommendation to use Agile methods like Scrum to deliver software on time and within budget. Although the results of this research isolated the software development portion of an IT program, it can be inferred that cost and schedule overruns can be prevented by using Scrum.

DoDI 5000.02 provides the detailed procedures that guide the operation of incrementally deployed software intensive programs.[73] Unlike DOD-STD-2176, DoDI 5000.02 reflects *how* to improve the effectiveness and efficiency of IT programs without limiting *what* a program must use to accomplish expected results. For AF IT programs like ECSS that adapt COTS software, the expectation is for deployment of the full capability to occur in multiple releases of new capability within 18 to 24 month increments.[74]

Agile methods like Scrum fit the DoDI 5000.02 model to plan/analyze, design, build, test, and release new capability in small, short increments. An important caution cited in the guidance is as follows:

> An important caution in using this model is that it can be structured so that the program is overwhelmed with frequent milestone or deployment decision points and associated approval reviews. To avoid this, multiple activities or build phases may be approved at any given milestone or decision point, subject to adequate planning, well-defined exit criteria, and demonstrated progress.[75]

With respect to the caution, this research recommends that AF senior leaders make an organizational commitment to develop tailored Scrum training for execution of an AF IT program. The tailored training should include how to create and manage self-directed teams within a DoD environment, as well as how to mitigate acquisition and contractual impediments like not having a full set of defined requirements before contract award, pricing arrangements, and small business opportunities. Additionally, definitions for what constitutes adequate planning and well-defined exit criteria should be established before wide-spread participation in the tailored Scrum training.

CONCLUSION

Agile methods have become extremely popular in the field of software development over the last decade. The most common Agile methodology, Scrum, was introduced in 1995 as an alternative to the Waterfall methodology. Scrum was often contrasted with Waterfall; however, this research has shown that Scrum was closely aligned with the five additional features that Royce, Waterfall's founder, identified in 1970.[76]

When the five additional features and iterative interaction between steps were added to Waterfall, AF IT programs like ECSS were able to mitigate risks associated with cost and

schedule overruns. In the case of ECSS, Scrum could have prevented cost and schedule overruns. With Scrum, future AF IT programs would not waste taxpayer dollars on software development methodologies with high failure rates. Scrum could contribute to critical capabilities like supporting the warfighter.

NOTES

(All notes appear in shortened form. For full details, see the entry in the BIBLIOGRAPHY)

[1] eLog21 Fact Sheet, 2008.
[2] Dunn, 6.
[3] ECSS Fact Sheet.
[4] Aronin, et. al, 2011, 1.
[5] ECSS Fact Sheet.
[6] Aronin, et. al., 2011, 1.
[7] *25 Point Implementation Plan to Reform Federal Information Technology Management,* 2010, 1.
[8] Royce, 1970, 331 – 335.
[9] Leffingwell, 2007, 17-27.
[10] Royce, 1970, 331 – 335.
[11] Royce, 1970, 331 – 335.
[12] Sutherland, 2011.
[13] Aronin, et. al., 2011, 19.
[14] Aronin, et. al., 2011, 1.
[15] Aronin, et. al., 2011, 1.
[16] Aronin, et. al., 2011, 1.
[17] Aronin, et. al., 2011, 1.
[18] Rico, 3.
[19] "Development Life Cycle Models", 2012.
[20] "Development Life Cycle Models", 2012.
[21] "Development Life Cycle Models", 2012.
[22] Elliott, 2004, 87.
[23] Royce, 1970,328 – 338.
[24] Royce, 1970, 328 – 338.
[25] Royce, 1970, 328 – 338.
[26] Royce, 1970, 329.
[27] Royce, 1970, 329.
[28] Royce, 1970, 329.
[29] Royce, 1970, 329.
[30] Royce, 1970, 329.
[31] DOD-STD-2167, 1985, 2.
[32] Leffingwell, 2007, 19.

[33] Defense Science Board Task Force, 1987, 1.
[34] Defense Science Board Task Force, 1987, 3.
[35] Defense Science Board Task Force, 1987, 11.
[36] Defense Science Board Task Force, 1987, 11.
[37] Highsmith, 2015.
[38] "Principles behind the Agile Manifesto".
[39] "Principles behind the Agile Manifesto".
[40] "Principles behind the Agile Manifesto".
[41] Highsmith, 2015.
[42] Vile, 2011.
[43] "The Agile Manifesto".
[44] Harb, et al., 2015, 39.
[45] GAO-12-681, 2012, 7.
[46] Schwaber and Sutherland, 2013, 4.
[47] Schwaber and Sutherland, 2013, 5.
[48] Schwaber and Sutherland, 2013, 8.
[49] Schwaber and Sutherland, 2013, 6.
[50] Schwaber and Sutherland, 2013, 6.
[51] Schwaber and Sutherland, 2013, 5-6.
[52] Schwaber and Sutherland, 2013, 5.
[53] Schwaber and Sutherland, 2013, 9.
[54] Schwaber and Sutherland, 2013, 9.
[55] Schwaber and Sutherland, 2013, 4.
[56] Schwaber and Sutherland, 2013, 8.
[57] Schwaber and Sutherland, 2013, 9.
[58] Schwaber and Sutherland, 2013, 10.
[59] Schwaber and Sutherland, 2013, 10.
[60] Schwaber and Sutherland, 2013, 11.
[61] Schwaber and Sutherland, 2013, 12.
[62] Royce, 1970, 329.
[63] Royce, 1970, 331 – 335.
[64] Royce, 1970, 335.
[65] Royce, 1970, 335.
[66] Royce, 1970, 335.
[67] Royce, 1970, 335.
[68] Royce, 1970, 335.
[69] Aronin, et. al., 2011, 24.
[70] Rosa and Bilbro, 2012.
[71] Professional Scrum Training, 2015.
[72] Professional Scrum Training, 2015.
[73] *Directive 5000.02*, 2015, 11.
[74] *Directive 5000.02*, 2015, 11.
[75] *Directive 5000.02*, 2015, 12.
[76] Royce, 1970, 328 – 338.

BIBLIOGRAPHY

Aronin, Benjamin S., Bailey, John W., Byun, Ji S., Davis, Gregory A., Wolfe, Cara L., Frazier, Thomas P., and Bronson, Patricia F., *Expeditionary Combat Support System: Root Cause Analysis*, (Alexandria, Virginia: Institute for Defense Analyses, October 2011), 1 – 29.

Defense Science Board Task Force, *Report of the Defense Science Board Task Force on Military Software*, (Washington, D.C.: Department of Defense, September 1987), 1-11.

"Development Life Cycle Models," *National Instruments* (June 2012), http://zone.ni.com/reference/en-XX/help/371361J-01/lvdevconcepts/lifecycle_models/ (accessed: September 16, 2015).

Dunn, Grover, "A New Global Vision for Transforming Logistics," *Air Force Journal of Logistics*, (Summer 2007): 6.

Elliott, Geoffrey, *Global Business Information Technology: an integrated systems approach*, (Boston, MA: Pearson Education, 2004), 87.

"Expeditionary Combat Support System (ECSS) Fact Sheet," (Creech Air Force Base), http://www.creech.af.mil/shared/.../AFD-060831-042[1].pdf.

"Expeditionary Logistics for the 21st Century (eLog21) Fact Sheet," *DAU*, (Defense Acquisition University, March 2008), https://acc.dau.mil/CommunityBrowser.aspx?id=32784.

Harb, Yousra, Noteboom, Cherie, and Surendra Sarnikar, "Evaluating Project Characteristics for Selecting the Best-fit Agile Software Development Methodology: A Teaching Case," *Journal of the Midweast Association for Information Systems*, Issue 1, Volume 2015 (January 2015): 39.

Highsmith, Jim, "History: The Agile Manifesto," *Agile Manifesto*, http://agilemanifesto.org/history.html (accessed September 17, 2015).

Leffingwell, Dean, *Scaling Software Agility Best Practices for Large Enterprises*, (Boston, MA: Pearson Education, Inc., 2007), 17-27.

Military Standard (DOD-STD-2167), *Defense System Software Development*, (Washington, D.C.: Department of Defense, 4 June 1985), 2.

Office of Management and Budget (OMB), *25 Point Implementation Plan to Reform Federal Information Technology Management*, (Washington, D.C.: Department of Defense, December 9, 2010), 1-35.

"Professional Scrum Training," *Scrum.org*, http:www.scrum.org/Courses (accessed October 15, 2015).

Rico, David F., *Short History of Software Methods*, http://www.davidfrico.com/rico04e.pdf, 1-23.

Rosa, Wilson and Bilbro, James, *COTS Integration Estimation: Enterprise Resource Planning Systems*, (2012 University of Southern California CSSE Annual Research Review, March 2012).

Royce, Winston W., "Managing The Development of Large Software Systems", *Proceedings, IEEE WESCON* (August 1970): 328 – 338.

Schwaber, Ken and Sutherland, Jeff, *The Definitive Guide to Scrum: The Rules of the Game*, (July 2013), http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf , 1-16.

Sutherland, Jeff, *Scrum: Jeff Sutherland Breaks Down the Structure of Scrum* (January 21, 2011), https://www.youtube.com/watch?v=O7cA1q0XwhE (accessed September 17, 2015).

"The Agile Manifesto", *Agile Manifesto,* http://agilemanifesto.org (accessed September 17, 2015).

US Department of Defense, *Directive 5000.02*, (Washington, DC: Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics (USD AT&L), January 2015), 1-154.

US Government Accountability Office (GAO). *Software Development: Effective Practices and Federal Challenges in Applying Agile Methods*, GAO-12-681 (Washington, DC: July 2012), 1-34.

Vile, Dale. "Agile Development – The reality behind the philosophy," *Freeform Dynamics* (June 2011), http://www.freeformdynamics.com/fullarticle.asp?aid=1347 (accessed January 31,2015).